ELSEVIER

Theoretical Computer Science 259 (2001) 549–575

Theoretical Computer Science

www.elsevier.com/locate/tcs

Monotone term decision lists ☆

David Guijarro[a,*,1], Víctor Lavín[a,2], Vijay Raghavan[b,3]

[a] *Department LSI, Universitat Politècnica de Catalunya, Jordi Girona Salgado, 1–3, Barcelona 08034, Spain*

[b] *Box 1679-B, Computer Science Department, Vanderbilt University, Nashville, TN 37235, USA*

Received May 1998; revised February 1999

Communicated by O. Watanabe

Abstract

We introduce a new representation class of Boolean functions – monotone term decision lists – which combines compact representation size with tractability of essential operations. We present many properties of the class which make it an attractive alternative to traditional universal representation classes such as DNF formulas or decision trees. We study the learnability of monotone term decision lists in the exact model of equivalence and membership queries. We show that, for any constant $k \geqslant 0$, k-term monotone decision lists are exactly and properly learnable with $n^{O(k)}$ membership queries in $n^{O(k^3)}$ time. We also show that $n^{\Omega(k)}$ membership queries are necessary for exact learning. In contrast, both k-term monotone decision lists ($k \geqslant 2$) and general monotone term decision lists are not learnable with equivalence queries alone. We also show that a subclass of monotone term decision lists (disj-MDL) is learnable with equivalence and membership queries, while neither type of query alone suffices.

1. Introduction

1.1. Motivation

Several representations of Boolean functions have been proposed and studied. Among these are disjunctive normal form (DNF) formulas, decision trees, branching programs,

☆ A preliminary version of this work was presented at Eurocolt'97 [15].

* Corresponding author.

E-mail addresses: david@lsi.upc.es (D. Guijarro), vlavin@lsi.upc.es (V. Lavín), raghavan@vuse.vanderbilt.edu (V. Raghavan).

[1] Supported by the Esprit EU BRA program under project 20244 (ALCOM-IT) and by the spanish DGES PB95-0787 (Koala).

[2] Supported by grant FP93 13717942 from the Spanish Government.

[3] Supported by NSF grant CCR-9510392. Thanks to the LSI department of UPC, where this work was done while on sabbatical leave.

0304-3975/01/$ - see front matter
PII: S0304-3975(00)00043-8

decision lists, circuits, and Boolean formulas [30]. All these representation classes are *universal* in their ability to represent all Boolean functions. By the same token, however, they are all susceptible to the "Shannon effect" [30]: a random Boolean function almost surely will *not* have a small representation size, regardless of the representation class. This invalidates any categorical judgment of what constitutes the ideal representation class. In practice, the representation class to be used is dictated by the use – an application involving the design of classical digital circuits would use Boolean circuits or formulas while certain kinds of learning applications would opt for decision tree classifiers. Aside from this, there is the matter of individual taste – an argument, not entirely tongue-in-cheek, has been made that humans find a DNF representation most comprehensible.

In spite of the inherent difficulties in deciding the right denominational hallmarks of a good representational class, most practitioners would accept certain pragmatic considerations as being among them. Representation size is important in most applications and *everything else being equal* one would prefer a representation which is succinct. From this standpoint, one would prefer DNF formulas to decision trees, arguing that every Boolean function can be represented as a DNF formula of size no larger than the corresponding decision tree. The catch is that everything else is typically not equal. Indeed decision trees are "better" than DNF from the point of view of tractability of commonly used operations: one can do equivalence testing, testing for tautology, rendering to an irredundant form, truth-table minimization, and counting the number of satisfying assignments in polynomial time while none of these operations can be done in polynomial time for DNF formulas, assuming $P \neq NP$.

Often a restriction of a general representation class is deemed to be interesting and more useful than the original. Arguably, the restriction of general branching programs to read-once branching programs is more useful for a lot of applications [9]. One has not lost the universality of representation, but some operations are now tractable. (For example, equivalence testing is in co-RP [7] and one can also count the number of satisfying assignments in polynomial time.) In a similar vein, restricted classes of Petri nets have been proposed [23], which enable one to decide certain kinds of reachability problems efficiently, problems which are important but which provably have exponential time and space lower bounds in general Petri nets [21]. In this paper, we study the restriction of decision lists to monotone term decision lists, for a similar motivation.

1.2. Decision lists and monotone term decision lists

Decision lists were introduced by Rivest [25]. Informally, they represent Boolean functions as follows (formal definitions are in the next section): A decision list contains a sequence of nodes. Each node comprised a term (i.e., a conjunction of literals) together with a classification of *True* or *False*. An assignment is evaluated by finding the *first* node which contains a term which accepts the assignment and then taking the classification of the node as the evaluation. (The last node of a decision list contains the empty term, enabling all assignments to be evaluated no further than the default last

node.) Rivest made a study of decision lists and showed that the class of k-decision lists is properly PAC-learnable in polynomial time, for constant $k \geqslant 0$. Here the constant k refers to the maximum number of literals in a term of the decision list.

It has not been easy to generalize Rivest's learning results to allow arbitrary number of literals in a term. Such a generalization would include all Boolean functions – moreover, the smallest decision list representation of a Boolean function is no bigger than the DNF representation. Therefore, predictability of general decision lists would imply the predictability of DNF formulas, a hard open problem. Also, one can prove that even with a fixed number of literals, decision lists are not amenable to efficient handling in post-learning applications. Questions such as "Do two decision lists represent the same function?" and "Is a given decision list irredundant?" are co-NP complete.

Simon [26] and Castro and Balcázar [11] have considered certain simplifications of the general PAC-learning problem to allow one to learn decision lists containing terms of arbitrary length. Nevertheless, the sheer hardness of most elementary operations on decision lists manifests itself in limiting the scope of fruitful extensions of Rivest's results.

In this paper, we take an entirely different approach – we restrict the terms to be monotone. We show that this is not at all a big restriction: all Boolean functions can still be represented with decision lists with monotone terms only. In fact, we argue below that the representation class of monotone term decision lists is very competitive when compared with other representation classes. We hope that the results presented in this paper will be extended by other researchers to resolve some of the open questions mentioned here.

1.3. Representation size and other issues

A significant advantage of monotone term decision lists is that they are amenable to basic operations on representations. We show that it is polynomial-time decidable if a given monotone term decision list is irredundant (i.e., if it contains no irrelevant variables in a term and no irrelevant term) and make it irredundant if it is not. We also show that there is a polynomial-time equivalence test for monotone term decision lists. In contrast, for general decision lists, DNF formulas, and read-k branching programs for $k > 1$ these operations cannot be done in polynomial time unless $\mathrm{P} = \mathrm{NP}$.

Monotone term decision lists share with decision lists (and branching programs) the property that the representation of the complement of a Boolean function is of the same size, and easily computable. This is not true for some other representations (for example, DNF formulas). Also, for any fixed constant k, a monotone term decision list representation of the disjunction or conjunction of k monotone term decision lists may be constructed in polynomial time. In particular, this already implies that monotone term decision lists are polynomially closed under finite exceptions [4]; however, we prove the stronger result that this class is strongly polynomially closed under finite exceptions lists [8].

Monotone term decision lists seem to be as good or better than decision trees in every respect. Both representations allow efficient manipulation for operations such as the ones described above. In addition, we show that the monotone term decision list representation of a Boolean function is never bigger than the corresponding decision tree representation. On the other hand, there exist families of functions whose decision tree representations are exponentially larger than the monotone term decision list representation.

A comparison of representation size with other universal classes does not yield a clear winner. (Excepting decision lists, of course: naturally, a general decision list representation is bound to be no larger than a monotone term decision list representation.) A DNF/CNF representation size is incomparable with the corresponding monotone term decision list representation – there are functions whose DNF (or CNF) representation size is exponential in the monotone term decision list size, and vice versa. Nevertheless, the size of a monotone term decision list is never more than a quasipolynomial factor away from the *larger* of the CNF or DNF representation.

1.4. Learning

Our main results in the learnability of monotone term decision lists are as follows. First, we show that k-term monotone decision lists are properly and exactly learnable in $n^{O(k^3)}$ time with $n^{O(k)}$ membership queries alone, while equivalence queries alone will not suffice. Our algorithm is *non-adaptive* (i.e., it always makes queries on a fixed set of assignments). From this it follows that k-term monotone decision lists are "simple-PAC" learnable in the sense of Li and Vitányi [20].

Next, we consider the general class of monotone term decision lists with unrestricted number of terms. Using Angluin's technique of approximate fingerprints [2], we show that equivalence queries alone do not suffice for exact learning this class. Moreover, it is easy to see that the class cannot be learned with membership queries alone. An open problem, not resolved in this paper, is whether the class is exactly learnable with equivalence and membership queries. We do show, however, that a large subclass of general monotone term decision lists is so learnable. This subclass, which is simultaneously a generalization of "read-once" monotone term decision lists and monotone functions, also has the characteristic of not being learnable with equivalence queries alone or with membership queries alone.

Other results in learning theory are related to ours. Since we show that k-term monotone decision lists are representable as k-decision lists, it follows that, for any constant k, k-term monotone decision lists are *improperly* PAC-learnable as k-decision lists using Rivest's algorithm [25]. A slight modification of the negative results in [16, 24] shows that proper PAC-learnability of this class without membership queries is not possible, unless RP = NP. Improper exact learning algorithms which use both membership and equivalence queries can also be obtained by using the techniques of Bshouty [10] or Kushilevitz [19] to show that $O(\log n)$-term monotone decision lists are learnable.

1.5. Organization

The rest of the paper is organized as follows. Section 2 presents definitions used along the paper. Section 3 contains properties of monotone term decision lists. Section 4 contains our results on learning k-term monotone decision lists. Section 5 contains results on the learnability of the general class.

2. Preliminaries

A *decision list* is an ordered list of pairs of Boolean functions. Each pair (t, o) in a decision list is called a *node* of the decision list, the function t being the *test* function and the function o the *output* function. The last node in a decision list is called the *default* node and has a constant test function that evaluates to 1 (true). The evaluation of a decision list $\langle (t_1, o_1), (t_2, o_2), \ldots \rangle$ on an assignment α is obtained by first finding the least i such that $t_i(\alpha) = 1$ and then outputting $o_i(\alpha)$. In the following sections, we say that assignment α *activates* the node (t_i, o_i), if i is the least integer such that $t_i(\alpha) = 1$.

A k-decision list [25] is a decision list in which all the test functions are monomials with at most k literals and the output functions are the constants 0 and 1. A *monotone term decision list* is a decision list in which all the test functions (or *terms*) are monotone monomials and the output functions are the constants 0 and 1. If the output function associated with a node is 0, we say that the node is *negative*; otherwise it is *positive*. In this paper, we focus on monotone term decision lists in general and k-term monotone decision lists in particular, where $k > 0$ is some fixed integer constant. A k-term monotone decision list has at most k nodes, not counting the default node.

A monotone term decision list is *minimal* (or *irredundant*) if no node or variable within a term can be deleted without changing the Boolean function represented by the decision list. Note that a minimal monotone term decision list may be bigger than the *minimum* monotone term decision list for the same function.

The set of variables that are set to 1 in an assignment α is denoted by $ones(\alpha)$ and the set of variables that are set to 0 by $zeroes(\alpha)$. The natural partial order $>$ over assignments, given by $\alpha > \beta$ if and only if $ones(\alpha) \supset ones(\beta)$, defines a lattice called the *Boolean lattice*. The set of variables of a term t is denoted by $vars(t)$. Finally, for any assignment α, the assignment $\alpha_{v \leftarrow b}$ is the assignment obtained by setting the variable v to b and all the other variables as in α.

Our learning results are in the concept learning framework where an algorithm is required to identify (exact learning) or approximate (PAC learning) an unknown concept via queries (in the exact model) or random examples (in the PAC model).

We mainly use the exact learning model with membership and/or equivalence queries. Let f be the unknown *target* Boolean function to be learned. A *membership* query, MQ(α), receives an assignment α and returns $f(\alpha)$. An *equivalence* query, EQ(H) receives as input some representation H of a Boolean function and returns a "Yes" answer if $H \equiv f$ or a counterexample α such that $H(\alpha) \neq f(\alpha)$. If H is in the representation class being learned we say that the algorithm is a *proper* learning algorithm.

We say that a representation class R is polynomial time learnable in the exact model with queries (membership, equivalence, or both) if there exists an algorithm which uses these queries and, for any Boolean function f representable in R, outputs a representation of f (in R if it is proper or in some other representation class if it is improper), in time polynomial in the number of variables and the smallest size needed to represent f in R. For more formal definitions we direct the reader to [1, 2, 17, 29].

The PAC model differs from the exact model in two ways: (1) the goal of the learning algorithm is to output a "good" approximation of the target concept with high probability and (2) the information about the target concept is received via labeled examples drawn according to an unknown but fixed distribution that is also used to measure the "goodness" of the approximation. An algorithm is a polynomial time PAC-learning algorithm for the representation class R if for any distribution D, for any target Boolean function f representable in R, and for any, $\varepsilon, \delta > 0$, the algorithm outputs a hypothesis h (in R, if it is required to be proper) such that

$$\text{Prob}(\text{Prob}_D(h \triangle f) > \varepsilon) < \delta,$$

in time polynomial in the number of variables and the smallest size needed to represent f in R, $1/\varepsilon$, and $1/\delta$. Again, for more formal definitions we direct the reader to [5, 18, 22, 28].

3. Properties of monotone term decision lists

3.1. Universality

We begin by showing that the class of monotone term decision lists has the power to represent all Boolean functions.

Proposition 1. *Every Boolean function can be represented as a monotone term decision list.*

Proof. Let f be any Boolean function. To represent f using a monotone term decision list, walk through the Boolean lattice of assignments in topological order (i.e., starting at the assignment of all 1's and ending at the assignment of all 0's.) For each assignment α, create a node $(t_\alpha, f(\alpha))$, where t_α is a monotone monomial containing precisely the variables in $ones(\alpha)$. (Note that the evaluation of f on the all 0's assignment will form the output of the default node.) The decision list thus constructed represents f since each assignment α in the Boolean hypercube activates the corresponding node $(t_\alpha, f(\alpha))$. □

3.2. Efficient tests for equivalence, satisfiability, irredundancy, and monotonicity

Now we show that there are polynomial time algorithms for testing the equivalence of two monotone term decision lists.

Proposition 2. *There is an* $O(n(p+q)pq)$ *time algorithm that tests equivalence of two monotone term decision lists on n variables,* L_1 *and* L_2*, with p and q nodes, respectively. The algorithm also supplies an assignment that witnesses the inequivalence, if* $L_1 \not\equiv L_2$.

Proof. For any pair of terms t_1 and t_2 from L_1 and L_2, respectively, define the assignment $\beta(t_1,t_2)$ by $ones(\beta(t_1,t_2)) = vars(t_1) \cup vars(t_2)$. Clearly, if there exists a term t_1 in L_1 and a term t_2 in L_2 such that $L_1(\beta(t_1,t_2)) \neq L_2(\beta(t_1,t_2))$, then $L_1 \not\equiv L_2$. Conversely, if $L_1 \not\equiv L_2$, then there exists some assignment α such that $L_1(\alpha) \neq L_2(\alpha)$. Assume that α activates nodes (t_1,b) in L_1 and $(t_2,\bar{b})$ in L_2. Consider the assignment $\beta = \beta(t_1,t_2)$ (observe that $\beta \leqslant \alpha$). Now β activates both (t_1,b) and $(t_2,\bar{b})$, and $L_1(\alpha) = L_1(\beta) \neq L_2(\beta) = L_2(\alpha)$.

From the above, it follows that testing L_1 and L_2 for equivalence is tantamount to checking if $L_1(\beta(t_1,t_2)) = L_2(\beta(t_1,t_2))$ for all pairs t_1 in L_1 and t_2 in L_2. There are $O(pq)$ pairs of terms to consider, and each evaluation takes $O(n(p+q))$ time. □

The algorithm for testing equivalence can be used to test if a monotone term decision list is irredundant: all one needs to do to decide if a particular variable in a term (or a particular node) is redundant is to delete the variable from the term (or the entire node) and test the resultant decision list for equivalence with the original. Similarly, we can test satisfiability/tautology of a given monotone term decision list by checking if an irredundant version is precisely the 0-term monotone decision list which represents false/true.

Deciding whether a monotone term decision list represents a monotone Boolean function is also straightforward: all one needs to do is first render the given list irredundant, and then test if all the non-default outputs are 1. This will happen precisely if the represented function is monotone since an irredundant monotone term decision list representing a monotone Boolean function must contain precisely the minterms of the function as its non-default terms.

Note that *none* of the tests mentioned in this subsection can be done in polynomial time for general decision lists if $P \neq NP$. (This can be derived by first observing that a Boolean function represented by a DNF or a CNF formula can be polynomially transformed into a decision list. Other than satisfiability (tautology) testing, which is hard for CNF but easy for DNF (easy for CNF but hard for DNF), all other tests are known to be hard for DNF and CNF representations. Therefore, all these tests are hard for decision lists.) However, decision trees also allow all these operations to be done in polynomial time.

An open question is whether monotone term decision lists can be *minimized* in polynomial time, instead of merely being rendered irredundant as above. To the best of our knowledge, minimization in polynomial time is also open for decision trees.

3.3. Efficient computation of Boolean operations

The fundamental Boolean operations – NOT, AND, and OR – can be done in polynomial time on functions represented as monotone term decision lists, as we now show.

Proposition 3. *A monotone term decision list representing the complement of a given monotone term decision list can be constructed in linear time.*

Proof. Given a decision list L representing a Boolean function f, a decision list L' representing $\overline{f}$ can be obtained by simply flipping the output value of every node in L. Clearly, every assignment activates corresponding nodes in L and L' and is evaluated in a complementary manner.

Proposition 4. *There is an* $O(nlm)$ *time algorithm for constructing a representation of the disjunction of two monotone term decision lists on n variables,* L_1 *and* L_2*, with l and m nodes, respectively.*

Proof. Let $L_1=(t_1,b_1)\cdots(t_l,b_l)$ and $L_2=(s_1,c_1)\cdots(s_m,c_m)$ where $t_l=s_m=\text{True}$ are the default terms. The following monotone term decision list

$$R=[B_1]\cdots[B_l]$$

will be a representation of the OR of L_1 and L_2. Each block $[B_i]$ is a sequence of nodes such that if $b_i=1$ then $B_i=(t_i,1)$ and if $b_i=0$ then

$$B_i=(t_is_1,c_1)\cdots(t_is_m,c_m).$$

Now let α be an assignment that activates a node in block $[B_i]$. Note that the last node in every block $[B_j]$ contains the term t_j, regardless of whether t_j evaluates to 0 or 1 in L_1. Therefore, we may conclude that α activates the node (t_i,b_i) in L_1. Furthermore, if α activates node (t_is_j,c_j) in block $[B_i]$ in R then α also activates node (s_j,c_j) in L_2 (otherwise α will activate an earlier node inside block $[B_i]$). Consequently, $R(\alpha)=0$ if and only if α activates some node $(t_is_j,0)$ in a block $[B_i]$, which happens if and only if α activates $(t_i,0)$ in L_1 and $(s_j,0)$ in L_2. In other words, R represents the disjunction of L_1 and L_2. □

From Propositions 3 and 4 and the fact that $f_1\cdot f_2=\overline{\overline{f_1}+\overline{f_2}}$, it follows that the conjunction of two monotone term decision lists can also be constructed efficiently. Moreover, these results imply that monotone term decision lists are polynomially closed under finite exceptions in the sense of Angluin and Kriķis [4]. In the following proposition, we prove a stronger result which implies that monotone term decision lists are strongly polynomially closed under exception lists (in the sense of Board and Pitt [8]).

Proposition 5. *Let L be a monotone term decision list of p nodes representing a function f over n variables. Let A be a set of assignments and f_A be the function defined by*

$$f_A(\alpha) = f(\alpha) \quad \text{if and only if} \quad \alpha \notin A.$$

Then, a monotone term decision list L_A of at most $p + n \cdot |A|$ nodes representing f_A can be constructed in $O(pn|A|)$ time.

Proof. It suffices to show that for a single assignment α, a decision list representing $f_{\{\alpha\}}$ can be constructed by adding at most n nodes to L in time $O(pn)$. To do this, let (t, b) be the node in L that is activated by α and insert, just in front of the node (t, b), the sequence of nodes

$$\langle (t_\alpha v_1, b),\ (t_\alpha v_2, b), \ldots, (t_\alpha v_l, b), (t_\alpha, \bar{b}) \rangle,$$

where $vars(t_\alpha) = ones(\alpha)$ and $zeroes(\alpha) = \{v_1, v_2, \ldots, v_l\}$. This can clearly be done in the claimed time. □

Proposition 5 implies that if there is a PAC-algorithm for learning monotone term decision lists properly then there exists a randomized polynomial-time Occam algorithm for monotone term decision lists [8]. Closure under finite exceptions implies that if monotone term decision lists are exactly learnable with equivalence and membership queries then they are also so learnable even with a small amount of malicious noise in membership queries [4].

3.4. Size comparisons with other classes

In this section we compare the representation size of monotone term decision lists with other universal classes: decision trees, DNF, CNF, and parities of Monotone DNFs.

We start with decision trees. A decision tree is a binary tree with variables in the internal nodes and the constants 0 and 1 in the leaves. The evaluation of an assignment starts in the root of the tree and consists of a path that leads to a leaf. The path is decided by the values that the assignment has in the variables tested in the internal nodes in the following way: at some internal node that contains the variable x, the path branches to the right child if the assignment contains a 1 in x and to the left child otherwise. The output value for an assignment corresponds to the value of the leaf of the evaluation path.

Let $|f|_{dt}$, $|f|_{mdl}$, $|f|_{dnf}$, and $|f|_{cnf}$ be, respectively, the number of leaves of the smallest decision tree, the number of nodes of the smallest monotone term decision list, the number of terms of the smallest DNF formula, and the number of clauses of the smallest CNF formula that represents f.

Theorem 6. *For any Boolean function f, $|f|_{dt} \geqslant |f|_{mdl}$.*

Proof. The proof is constructive. Given a decision tree we will show how to construct a monotone term decision list that computes the same function and has as many nodes as leaves in the decision tree. For a complete path p, from the root to a leaf, we define a node (t_p, o_p) where t_p is the term formed by the conjunction of all right-branching variables of p and o_p is the output value of the leaf of p. Now consider the monotone term decision list $T = \langle (t_{p_1}, o_{p_1}), (t_{p_2}, o_{p_2}), \ldots, (t_{p_r}, o_{p_r}) \rangle$, where the paths $p_1, p_2, \ldots, p_r$ are in the order obtained by placing the corresponding leaves according to the reverse of an inorder visit, i.e., starting at the rightmost one and ending at the leftmost one.

To see that the decision tree and the decision list represent the same Boolean function, observe that an assignment is evaluated at some path p in the decision tree if and only if it activates the node (t_p, o_p) in the monotone term decision list. □

Theorem 7. *There exists a family of Boolean functions* $\{f_n\}_{n>0}$ *such that both* $|f_n|_{dnf}$ *and* $|f_n|_{cnf}$ *are exponential in* $|f_n|_{mdl}$.

Proof. Let $V_n = \{x_1, x_2, \ldots, x_{2n}\} \cup \{y_1, y_2, \ldots, y_{2n}\}$ be a set of $4n$ Boolean variables. Define a monotone function $m_n = y_1 y_2 + y_3 y_4 + \cdots + y_{2n-1} y_{2n}$ and an antimonotone function $a_n = (\bar{x}_1 + \bar{x}_2)(\bar{x}_3 + \bar{x}_4) \ldots (\bar{x}_{2n-1} + \bar{x}_{2n})$. Let $f_n = m_n \cdot a_n$.

Consider the following monotone term decision list:

$$M_n = \langle (x_1 x_2, 0), (x_3 x_4, 0), \ldots, (x_{2n-1} x_{2n}, 0), (y_1 y_2, 1), \\ (y_3 y_4, 1), \ldots, (y_{2n-1} y_{2n}, 1)(True, 0) \rangle.$$

M_n is satisfied by the assignments which set at least one of x_{2i-1} or x_{2i} to false for all i, $1 \leqslant i \leqslant n$, and both y_{2j-1} and y_{2j} to true, for some j, $1 \leqslant j \leqslant n$. These are precisely the assignments which will satisfy both a_n and m_n – in other words, $M_n \equiv f_n$. So f_n can be represented as a monotone term decision list of $2n$ terms.

Next, consider the set of assignments

$$P = \{\alpha \mid \alpha \text{ sets precisely one of } x_{2i-1} \text{ and } x_{2i} \text{ to } 0,\ \forall i, 1 \leqslant i \leqslant n, \\ \text{and both } y_{2j-1} \text{ and } y_{2j} \text{ to } 1, \text{ for exactly one } j,\ 1 \leqslant j \leqslant n\}.$$

All assignments in P are positive assignments of f_n. Moreover, no two assignments in P can be accepted by any implicant of a DNF formula equivalent to f_n, lest such as implicant also accepts a negative assignment of f. Consequently, each assignment in P must be accepted by a different term of a DNF formula representing f_n, or $|f_n|_{dnf} \geqslant |P| = n2^n$.

A dual argument proves that $|f_n|_{cnf} \geqslant 2^n$. □

Using the well-known fact that $|f|_{dt} \geqslant |f|_{dnf} + |f|_{cnf}$, we get the following corollary:

Corollary 8. *There exists a family of Boolean functions* $\{f_n\}$ *such that* $|f_n|_{dt}$ *is exponential in* $|f_n|_{mdl}$.

A result implicit in the work of Ehrenfeucht and Haussler [12] and Fredman and Khachiyan [13], is that $|f|_{dt} \leqslant |f|_{cdnf}^{\log^2 |f|_{cdnf}}$, where $|f|_{cdnf} = \max\{|f|_{dnf}, |f|_{cnf}\}$. Using Theorem 6 and this result leads to the following corollary.

Corollary 9. *For any Boolean function* f, $|f|_{mdl} \leqslant |f|_{cdnf}^{\log^2 |f|_{cdnf}}$.

However, the following result shows that it is possible for the size of the smallest monotone term decision list representing a Boolean function to be exponential in the *minimum* of the DNF and CNF size of that function.

Theorem 10. *There exist families of functions* $\{f_n\}$ *and* $\{g_n\}$ *such that* $|f_n|_{mdl}$ *is exponential in* $|f_n|_{dnf}$ *and* $|g_n|_{mdl}$ *is exponential in* $|g_n|_{cnf}$.

Proof. If f is a monotone function, then any monotone term decision list representation of f must have a node for each prime implicant of f. Similarly, if f is an antimonotone function, then any monotone term decision list representation of f must have a node for each prime clause of f. The monotone n-clause CNF formula $g_n = (x_1 + x_2)(x_3 + x_4)\cdots(x_{2n-1} + x_{2n})$ has 2^n prime implicants and the antimonotone n-term DNF formula $f_n = \bar{x}_1\bar{x}_2 + \bar{x}_3\bar{x}_4 + \cdots + \bar{x}_{2n-1}\bar{x}_{2n}$ has 2^n prime clauses. Therefore, $|f_n|_{mdl}$ and $|g_n|_{mdl}$ are at least 2^n.

Takimoto et al. [27] considered the learnability of a representation class based on the exclusive or of a sequence of monotone DNF formulas, each one of which implies the next one in the sequence. More precisely, each function in the class $\oplus$MDNF, is a representation of the form

$$f = f_1 \oplus f_2 \oplus \cdots \oplus f_s,$$

where each f_i, $1 \leqslant i \leqslant s$, is a monotone DNF formula and $f_i \Rightarrow f_{i+1}, 1 \leqslant i < s$. As we show below, this representation class is very closely related to monotone term decision lists – given a $\oplus$MDNF representation of f, a monotone term decision list representation can be obtained in polynomial time, and vice versa.

Let $|f|_{\oplus mdnf}$ denote the fewest total number of terms required to represent f as a $\oplus$MDNF formula.

Theorem 11. *For any Boolean function* f

1. $|f|_{mdl} \leqslant |f|_{\oplus mdnf}$
2. $|f|_{\oplus mdnf} = \mathrm{O}(|f|_{mdl}^2)$

Moreover, given either a monotone term decision list or a $\oplus$*MDNF representation of* f, *one can construct the other representation in polynomial time.*

Proof. To prove 1, let $f_1 \oplus f_2 \oplus \cdots \oplus f_s$ be a $\oplus$MDNF representation of a Boolean function f over n variables. Using the fact that $f_i \Rightarrow f_{i+1}, 1 \leqslant i < s$, one can prove the

following fact using induction on s:

- If s is odd, then $f = f_1 + \overline{f}_1\overline{f}_2\overline{f}_3 + \cdots + \overline{f}_1\overline{f}_2\overline{f}_3 \ldots \overline{f}_{s-1}f_s$.
- If s is even, then $f = \overline{f}_1 f_2 + \overline{f}_1\overline{f}_2\overline{f}_3\overline{f}_4 + \cdots + \overline{f}_1\overline{f}_2\overline{f}_3 \ldots \overline{f}_{s-1}f_s$.

Construct a monotone term decision list consisting of blocks of nodes $B_1, B_2, \ldots B_s$, where block B_i contain precisely the terms of f_i in its nodes and all the nodes have the same output value o_i, determined by the following rule. Fix the output value for the last block, o_s, to be 1, the value for the previous block, o_{s-1}, to be 0, and so on, alternating between 1 and 0 until the first block is reached. Finally, add a default node $(True, 0)$. The above fact implies that the monotone term decision list so constructed represents the same function. Clearly, this decision list can be constructed in $O(sn)$ time.

To prove 2, let L be an irredundant monotone term decision representation of f. Without loss of generality, assume that the default node is $(True,0)$. (If not, *add* this node as an unused default.) Note that the node before the default node must have an output value of 1. Define a *block* of nodes of L to be a maximal sequence of consecutive nodes of L, all of which have the same output value. Let $B_1, B_2, \ldots, B_s$ be the blocks of L, where B_s contains nodes up to but not including the default node $(True,0)$. Let $g_1, g_2, \ldots, g_s$ be monotone DNF formulas where g_i contains all the monotone terms of block B_i. Now, by the definition of blocks, it is easy to see that:

- If s is odd, then $f = g_1 + \overline{g}_1\overline{g}_2\overline{g}_3 + \cdots + \overline{g}_1\overline{g}_2\overline{g}_3 \ldots \overline{g}_{s-1}g_s$.
- If s is even, then $f = \overline{g}_1 g_2 + \overline{g}_1\overline{g}_2\overline{g}_3 g_4 + \cdots + \overline{g}_1\overline{g}_2\overline{g}_3 \ldots \overline{g}_{s-1}g_s$.

Finally, define s monotone DNF formulas by $f_1 = g_1$ and in general, $f_i = f_{i-1} + g_i$, $1 < i \leqslant s$. Clearly, $f_i \Rightarrow f_{i+1}$, $1 \leqslant i < s$. Finally, an argument based on induction on s and the above fact shows that $f = f_1 \oplus f_2 \oplus \cdots \oplus f_s$. The resultant $\oplus$MDNF formula, even if the intermediate monotone term DNF formulas f_i are not rendered irredundant, contains no more than the square of the number of non-default nodes in the decision list. The whole construction can be achieved in $O(s^2 n)$ time. □

The above result implies that monotone term decision lists are learnable in polynomial time if and only if $\oplus$MDNF formulas are so learnable, in whatever model one chooses. In fact, virtually any algorithm to solve problems for one representation class transforms into an algorithm for the other class. Given this, the learning results in [27] are particularly interesting. The authors claim a proper polynomial time algorithm for $\oplus$MDNF using subset and superset queries. This result is based on the assertion that a $\oplus$MDNF representation has a canonical representation which has minimum size but, unfortunately this assertion does not hold in general: the "canonical" representation given in the paper for the formula $f_n(x_1, \ldots, x_n) = x_1 \oplus (x_1 \vee x_2) \oplus \cdots \oplus (x_1 \vee \cdots \vee x_n)$ has size exponential in n. This invalidates the claimed learning results for the class $\oplus$MDNF and consequently, their application to monotone term decision lists. Nevertheless, we hope that the ideas along the lines explored in [27] will yield positive results.

4. Learning k-term monotone decision lists

We consider the learnability of the class of k-term monotone decision lists for some constant $k \geqslant 0$. This class is a natural candidate to begin the study of learnability of monotone term decision lists since it is the smallest interesting subclass which allows arbitrary length terms. Our main result for this class is in Section 4.1 where we show that this class can be learned properly using only membership queries. In Section 4.2 we give lower bounds on the learnability of this class and show that the class is not learnable with equivalence queries alone. To finish off the section, we comment on PAC-learnability and improper learnability of the class in Section 4.3.

4.1. Proper exact learning

A *specification set* [14, 6] for a class C of Boolean functions is a set X of assignments such that no two functions in the class C evaluate all the assignments in X in exactly the same way. Clearly, if X is small and can be found efficiently, then learning of the class C can be achieved by making membership queries on X and then using the results to determine the target function.

Let A_l denote the set of assignments with at most l zeroes.

Proposition 12. *The set A_{2k} is a specification set for k-term monotone decision lists.*

Proof. Let L_1 and L_2 be k-term monotone decision lists. We need to show that $L_1 \not\equiv L_2$ if and only if for all assignments $\alpha \in A_{2k}$, $L_1(\alpha) \neq L_2(\alpha)$.

The "only if" part is easy. If there exists an $\alpha \in A_{2k}$ such that $L_1(\alpha) \neq L_2(\alpha)$ then certainly $L_1 \not\equiv L_2$.

For the "if" part, suppose that α is an assignment such that $L_1(\alpha) \neq L_2(\alpha)$. In particular, let (t_1, b) and $(t_2, \overline{b})$ be the respective nodes of L_1 and L_2 activated by α. Let T be the set of terms that precede t_1 in L_1 or t_2 in L_2. Note that $ones(\alpha) \supseteq vars(t_1) \cup vars(t_2)$, and $zeroes(\alpha)$ must contain at least one variable from each term in T. This implies that an assignment α' in which $zeroes(\alpha')$ is a minimal subset of $zeroes(\alpha)$ which contains at least one variable from each term in T will also satisfy $L_1(\alpha') \neq L_2(\alpha')$. Such an assignment α' must be in A_{2k} since $|T| \leqslant 2k$. □

Proposition 12 implies that k-term monotone decision lists can be learned using $|A_{2k}| = \sum_{i=0}^{2k} \binom{n}{i} \leqslant (en/2k)^{2k}$ membership queries alone. However, it is not clear how one can infer *in polynomial time* the target k-term monotone decision list, knowing its evaluation on the assignments in A_{2k}. Indeed, it is not even clear whether the evaluations on A_{2k} can be used to predict in polynomial time the evaluations of the target on all assignments.

We address these issues now.

Proposition 13. *Let f be a Boolean function representable as a k-term monotone decision list, α be any assignment, and $b \in \{0, 1\}$. Then, $f(\alpha) = b$ if and only if there*

exists some set $S \subseteq zeroes(\alpha)$ *of at most* k *variables, such that for all assignments* $\beta \in A_{2k}$ *that satisfy zeroes* $(\beta) \supseteq S$ *and ones* $(\beta) \supseteq ones(\alpha)$, $f(\beta) = b$.

Proof. Assume that α activates the ith node (t_i, b) of some k-term monotone decision list representation L of f. For any pairwise disjoint sets Y and Z of variables, let $p(Y,Z)$ denote the partial assignment that sets to 0 all the variables in Y and to 1 the variables in Z.

To prove the "only if" part, let $S \subseteq zeroes(\alpha)$ be a set of at most $i-1$ variables whose negation falsify the first $i-1$ terms of L. Now consider the partial assignment $p(S, ones(\alpha))$. Clearly, all extensions of $p(S, ones(\alpha))$ – in particular those that contain at most $2k$ 0's – activate (t_i, b), and are therefore classified as b.

Conversely, for any assignment α suppose there exists a set S as in the proposition. Suppose, for contradiction, that α activates the node $(t_i, \bar{b})$ in some k-term monotone decision list representation L of f. Now consider the assignment β that sets to 0 the variables in S, sets to 0 one variable in $vars(t) - (vars(t_i) \cup ones(\alpha))$ for each term t in L before t_i that does not contain any variable of S, and sets to 1 the remaining variables. Clearly, β is in A_{2k} and is an extension of $p(S, ones(\alpha))$; however, β activates $(t_i, \bar{b})$, which contradicts the supposition that every extension of $p(S, ones(\alpha))$ evaluates to b. □

The following corollary is immediate.

Corollary 14. *Let* f *be a Boolean function representable as a* k*-term monotone decision list. Given* $f(\beta)$ *for each assignment* $\beta \in A_{2k}$, *we can determine* $f(\alpha)$ *for any assignment* α *in* $n^{O(k)}$ *time.*

Corollary 14 implies that k-term monotone decision lists can be *improperly* exactly learned using membership queries alone. To derive a proper learning algorithm, one must use the evaluations on A_{2k} to find a k-term monotone decision list representation. The following proposition provides the key idea. We begin with a definition.

Definition 15. Let $L = [(t_1, b_1) \cdots (t_l, b_l)(True, \bar{b}_l)]$ be any monotone term decision list. We say that an assignment α *jumps over* the node (t_i, b_i) in L if $L(\alpha) = \bar{b}_i$ and for all $j \leqslant i$, $t_j(\alpha) = 0$.

Proposition 16. *Let* $L = [(t_1, b_1) \cdots (t_k, b_k)(True, \bar{b}_k)]$ *be any minimal monotone term decision list.*

(a) *Let* (t_i, b_i) *be a node in* L *and let* v *be any variable in* t_i. *Then there exists a node* $(t_j, \bar{b}_i)$ *with* $j > i$ *and an assignment* $\alpha_{i,v}$ *defined by ones* $(\alpha_{i,v}) = (vars(t_i) \cup vars(t_j)) - \{v\}$, *such that* $\alpha_{i,v}$ *activates the node* $(t_j, \bar{b}_i)$ *in* L.

(b) *There exist sets* $S_1, \ldots, S_k$ *such that for each* i, $1 \leqslant i \leqslant k$,
   (i) $\forall v,\ v \in vars(t_i) \Leftrightarrow (\exists \beta \in S_i\ (\beta_{v \leftarrow 0}$ jumps over $(t_i, b_i)))$,
   (ii) $S_i \subseteq A_k$, *and*
   (iii) $|S_i| \leqslant k - i$ *if* $i \neq k$ *and* $|S_k| = 1$.

Proof of (a)**.** Let L' be the monotone term decision list obtained from L by deleting the variable v from the term t_i. By the minimality of L, L' computes a different Boolean function and hence there is an assignment α such that $L(\alpha) \neq L'(\alpha)$ which is a counterexample to the equivalence of L and L'.

Now α must activate the new node $(t_i - \{v\}, b_i)$ in L' for it to be a counterexample. (Note that any assignment which activates nodes earlier or later than the new node in L' must activate the same node in L.) Also, α cannot activate a node earlier than (t_i, b_i) in L lest it activate the same node in L'. Therefore, some node $(t_j, \overline{b}_i)$ in L with $j > i$ must be activated by α.

To finish the proof, observe that setting to 0 all the variables in $ones(\alpha)$ that are not in $(vars(t_i) \cup vars(t_j)) - \{v\}$ still yields a counterexample to the equivalence of L and L'. □

Proof of (b)**.** First, consider the sets S_i', $1 \leqslant i \leqslant k$, defined by

$$S_i' = \{\beta\colon \exists j > i \text{ and } (t_j, \overline{b}_i) \in L,\ \exists v \in vars(t_i)\ (ones(\beta) = vars(t_i) \cup vars(t_j) \text{ and } \beta_{v \leftarrow 0} \text{ activates } (t_j, \overline{b}_i))\}.$$

Clearly, the sets S_i' satisfy requirement (iii). By part (a) of the proposition, for each variable v in $vars(t_i)$, there is at least one assignment $\beta \in S_i$ such that $\beta_{v \leftarrow 0}$ jumps over (t_i, b_i). Moreover, if a variable $v \notin vars(t_i)$ then for any $\beta \in S_i'$, $\beta_{v \leftarrow 0}$ activates some node no later than (t_i, b_i) in L – hence $\beta_{v \leftarrow 0}$ cannot jump over (t_i, b_i). Therefore, the sets S_i' satisfy the requirement (i) as well.

Now we show that the sets S_i' can be modified to satisfy the requirement (ii) too. To do this, replace each assignment $\beta' \in S_i'$ with exactly one assignment $\beta \in A_k$ such that β *satisfies precisely the same set of terms in* L *that are satisfied by* β'. This can be done since at most k zeroes are needed to falsify any set of terms of L. Then $\beta_{v \leftarrow 0}$ jumps over (t_i, b_i) if and only if $\beta'_{v \leftarrow 0}$ does. Consequently, the modified sets satisfy all three requirements. □

Given sets $S_1, \ldots, S_k$ and Boolean values $b_1, \ldots, b_k$ one can construct, in time polynomial in n and k, a monotone term decision list

$$L = L(S_1, \ldots, S_k, b_1, \ldots, b_k),$$

which satisfies Proposition 16(b)(i). Note that the default node of L is being implicitly specified as $(True, \overline{b}_k)$. Also note that such a list may not be minimal; however, using the results in Section 3.2, one can test for minimality efficiently.

This leads to the following algorithm for proper learning:

Algorithm Learn_k_term_monotone_decision_lists

1. Make membership queries on all assignments in A_{2k}.
2. For each, l, $1 \leqslant l \leqslant k$, generate all sequences of sets $S_1, \ldots, S_l$ and all sequence of Boolean values $b_1, \ldots, b_l$ such that:

- $S_i \subset A_l$, $1 \leqslant i \leqslant l$ and
- $|S_i| \leqslant l$, $1 \leqslant i \leqslant l$.

3. For each sequence of sets $S_1, \ldots, S_l$ and each sequence of Boolean values $b_1, \ldots, b_l$ generated, construct a candidate hypothesis

$$L = L(S_1, \ldots, S_l, b_1, \ldots, b_l)$$

and test if L is minimal and if for all $\alpha \in A_{2k}$, $L(\alpha)$ is the same as the evaluation returned in step (1). Stop when a candidate hypothesis which passes the test is found.

Correctness: Since we consider all possible sequences of sets $S_1, \ldots, S_l$ and Boolean values $b_1, \ldots, b_l$, $1 \leqslant l \leqslant k$, by Proposition 16 one of them must produce a minimal representation of the target. By Proposition 12, A_{2k} is a specification set for k-term monotone decision lists and so only a minimal candidate hypothesis equivalent to the target will pass the test in step 3.

Query complexity: Only membership queries are used and then only in the first step. The number of queries is $|A_{2k}|$, which is at most $(en/2k)^{2k} = n^{O(k)}$.

Time complexity: Up to a polynomial in n, the time complexity of the entire algorithm is bounded by the number of different sequences of sets $S_1, \ldots, S_l$ and Boolean values $b_1, \ldots, b_l$, $1 \leqslant l \leqslant k$, considered by the algorithm. This, in turn, is bounded by $\sum_{l=1}^{k} [\binom{|A_l|}{l}]^l \cdot 2^l < k \cdot \binom{n}{k}^k \cdot 2^k \in n^{O(k^3)}$. Therefore the overall time is in $n^{O(k^3)}$.

The comments on correctness and complexity have effectively proved the following theorem.

Theorem 17. *The class of k-term monotone decision lists can be properly learned with $n^{O(k)}$ membership queries in $n^{O(k^3)}$ time.*

The learning algorithm is non-adaptive, i.e., it uses membership queries only on the *fixed* set A_{2k}, independent of the target k-term monotone decision list. Therefore, it follows that k-term monotone decision lists are learnable in the "simple-PAC" model of Li and Vitányi [20].

4.2. Lower bounds

The results of the previous section are fairly tight in the sense that $n^{\Omega(k)}$ membership queries are also necessary for exact learning, for any constant k.

Theorem 18. *Any membership-query learning algorithm for k-term monotone decision lists must use at least $\sum_{i=1}^{k-1} \binom{n}{i} - 1$ queries.*

Proof. Let α be any assignment with at most $k-1$ zeroes. The singleton Boolean function which is true precisely on the assignment α can be represented as a k-term monotone decision list. To construct such a representation, first create nodes $(v, 0)$ for each variable v in $zeroes(\alpha)$, thus using at most $k-1$ nodes. Next, create a node $(t, 1)$, where $vars(t) = ones(\alpha)$, and finally set the default to 0.

From the above, it follows that the number of k-term monotone decision lists that can represent singletons is at least $\sum_{i=0}^{k-1}\binom{n}{i}$. An adversary who simply replies *false* for every query posed by a membership query based algorithm will force at least $\sum_{i=1}^{k-1}\binom{n}{i}-1$ queries, in order to rule out all but one singleton. □

We now turn to exact proper learnability with equivalence queries alone.

The class of 1-term monotone decision lists can be learned with equivalence queries alone as follows. Run two algorithms in pseudo-parallel – the first one will propose only hypotheses with 1 as the default classification, and the second only hypotheses with 0 as the default classification. One of the two algorithms will halt with the correct answer. Both algorithms start with a 1-term monotone decision list hypothesis made up of a single node having a monotone term containing all the variables and output value equal to the complement of the default. At each stage, if a counterexample α is received for the current hypothesis, then all the variables in $zeroes(\alpha)$ are deleted from the single term in the hypothesis. It is easy to see that one of the algorithms must halt with the correct output after at most n steps since at least 1 variable is deleted at each step.

For $k \geqslant 2$, k-term decision lists are not learnable in polynomial time with equivalence queries alone. To prove this, we show that the class of k-term monotone decision lists has a combinatorial property called *approximate fingerprints*. Due to a theorem of Angluin [2], the existence of this property suffices to show non-learnability of a class with equivalence queries alone.

Let $\mathscr{C}$ be a representation class of Boolean concepts, $\mathscr{C}_{m,n} \subseteq \mathscr{C}$ be the subclass containing representations of size at most m over n variables, and $\mathscr{C}_n$ be the subclass containing representations over n variables. That is, $\mathscr{C}_n = \bigcup_{m \geqslant 1} \mathscr{C}_{m,n}$ and $\mathscr{C} = \bigcup_{n \geqslant 1} \mathscr{C}_n$. We say that $\mathscr{C}$ has *approximate fingerprints* if, for every polynomial $p(m,n)$, there is some n_0, such that for all $n > n_0$, there is an $m = m(n)$ and a "target" representation class $T_{m,n} \subseteq \mathscr{C}_{m,n}$, such that for all $h \in \mathscr{C}_{p(m,n),n}$, there is some assignment α_h, such that fewer than $|T_{m,n}|/p(m,n)$ concepts in $T_{m,n}$ evaluate α_h the same way that h does.[4] Note that $|T_{m,n}|$ counts the actual number of concepts, not the number of representations.

Intuitively, a representation class which has approximate fngerprints cannot be learned with a polynomial number of equivalence queries because each possible hypothesis queried by a purported learning algorithm can be replied to with an "uninformative" counterexample which eliminates from consideration only a superpolynomially small fraction of the target class as candidates for the unknown concept to be learned.

We now show that the class of k-term monotone decision lists has approximate fingerprints for $k \geqslant 2$. We start with a lemma.

Lemma 19. *Let L be a k-term monotone decision list, $k \geqslant 2$. Then there exists an assignment $\alpha = \alpha(L)$ such that either*

(a) *α has at most $k-1$ zeroes, and $L(\alpha) = 0$, or*

(b) *α has at most n/k zeroes, and $L(\alpha) = 1$.*

[4] This is a simplification to Boolean representation classes of Angluin's general definition.

Proof. If L is the constant function 1 then (b) holds. Otherwise, if L is not a monotone Boolean function, or L has fewer than k terms, or if two terms in L share a variable, it is easy to construct an assignment that satisfies (a). Otherwise, all the k terms in L have distinct set of variables and L is a monotone Boolean function. Therefore, there must be a positive term in L with at most n/k variables. An assignment with ones exactly in the positions of the variables in such a term satisfies (b). □

Theorem 20. *For $k \geqslant 2$, the class of k-term monotone decision lists has approximate fingerprints.*

Proof. Let $T_{n,k}$ be the target class of k-term monotone decision lists in which all the terms are positive, the default is negative, and each term has exactly n/k variables that do not occur in any other term. To avoid floors and ceilings, assume that n is a multiple of k.

In order to prove the approximate fingerprints property, it suffices to show that the number of functions in $T_{n,k}$ that either classify an assignment of at most $k-1$ 0's as 0 or classify an assignment with at most n/k 1's as 1, is a superpolynomially small fraction of $|T_{n,k}|$.

Note that $|T_{n,k}| = (n!/[(n/k)!]^k k!)$. No function in $T_{n,k}$ classifies an assignment with at most $k-1$ zeroes as 0. Moreover, the fraction of the functions in $T_{n,k}$ that classify an assignment with at most n/k ones as 1 is at most

$$\frac{|T_{n-n/k,k-1}|}{|T_{n,k}|} = \frac{(n-n/k)!(n/k)!k}{n!} = \frac{k}{\binom{n}{n/k}},$$

which, after a routine use of Stirling's approximation, can be seen to be superpolynomially small in n for $k \geqslant 2$. □

Corollary 21. *The class of k-term monotone decision lists, $k \geqslant 2$, is not learnable with equivalence queries alone.*

Proof. Follows from Theorem 20 and Angluin's theorem [2] on the approximate fingerprints property. □

4.3. PAC-learning, improper learning, and related results

A paper [16] by Hancock et al. shows that k-term decision lists cannot be properly PAC-learned (without membership queries) unless RP = NP. The proof of this result can be altered slightly to show that k-term *monotone* decision lists cannot be properly PAC-learned either, under the same assumption. Nevertheless, as a consequence of the following proposition, k-term decision lists (monotone or not) can be *improperly* PAC-learned as k-decision lists using Rivest's algorithm [25].

Proposition 22. *Every k-term list can be represented as a k-decision list.*

Proof. Let L be a k-term decision list. We transform L to an equivalent k-decision list L' as follows: Initially L' is the empty list. We begin by adding to L' a set of nodes that classify correctly the assignments that activate the default in L, i.e., we add to L' nodes (t,b), where b is the classification of the default of L, and each term t is obtained by picking one literal from each (non-default) term of L and complementing these at most k literals. Next, for each node from the last node of L down to the second node of L, we use a similar procedure to classify all assignments that activate the node. That is, if (t_i,b) is the ith node of L, we add to L' one node (t,b) for every term t obtained by picking one literal from each of the $i-1$ terms of L prior to (t_i,b) and complementing the at most $i-1$ literals. Finally, we set the default of L' to be the classification of the first node of L. □

Improper exact learning of k-term monotone decision lists using membership queries only can also be obtained by using the algorithm in [10]. Furthermore, using the techniques of Bshouty [10] and Kushilevitz [19], one can improperly learn up to $O(\log n)$-term decision lists exactly by using membership *and* equivalence queries.

5. Learning general monotone term decision lists

The central question of whether the general class of monotone term decision lists is learnable, properly or not, in any model of learning is open. Here, we give some partial results. First of all, it follows from Theorem 18 that general decision lists are not learnable with membership queries alone, since all 2^n singletons are representable as n-term monotone decision lists. In Section 5.1 below, we show that equivalence queries alone also do not suffice for learnability. We close the section by showing that a large subclass of the general class is indeed learnable, even though it is not learnable with membership or equivalence queries alone.

5.1. Approximate fingerprints

We now show that monotone term decision lists have approximate finger prints. Note that this result does not follow immediately from the earlier proof of approximate fingerprints. We begin with a lemma.

Lemma 23. *Let L be a p-term monotone term decision list on n variables. There is an assignment $\alpha=\alpha(L)$ such that either*
(a) $L(\alpha)=0$ *and α contains at most $\sqrt{n\ln p}$ 0's, or*
(b) $L(\alpha)=1$ *and α contains at most $\sqrt{n\ln p}$ 1's.*

Proof. Let $m=\sqrt{n\ln p}$. If the first term in L is negative, then clearly the all 1's assignment satisfies (a). If the default is positive, then the all 0's assignment satisfies (b). Otherwise, let $p'\leqslant p$ be the number of positive terms in L. If any such positive term, say t, has at most m variables, then the assignment α such that $ones(\alpha)=vars(t)$

satisfies (b). Finally, if all the positive terms of L have more than m variables, then we show the existence of an assignment α that satisfies (a) as follows:

Pick $r=m$ variables at random and set them to 0, setting the remaining variables to 1. The probability that such an assignment is accepted by some positive term of L is at most

$$(1-m/n)^r p' \leqslant \left(1-\sqrt{\frac{\ln p}{n}}\right)^r p < \mathrm{e}^{-r\sqrt{(\ln p/n)}}\, p.$$

For our choice of r, the last quantity in the above expression is 1. Therefore, there exists some set X of at most r variables such that the assignment α formed by setting all the variables in X to 0 and the remaining variables to 1 is not accepted by any positive term of L. Such an assignment satisfies (a). □

Theorem 24. *The class of monotone term decision lists has approximate fingerprints.*

Proof. Let T_n be the class of $q=\sqrt{n}$-term monotone decision lists in which all the terms are positive, the default is negative, and each term has exactly q variables which do not occur in any other term. (Assume that q is an integer in order to avoid floors and ceilings.) Note that this is the same target class as the one used in [3] to show that read-once formulas cannot be identified with equivalence queries alone. The number of logically distinct Boolean functions in T_n is $n!/((q!)^{q+1})$.

Let $p=n^c$ for some fixed constant $c>0$ and let L be any monotone decision list of p terms. Let $\alpha=\alpha(L)$ be the assignment whose existence is guaranteed by Lemma 23. To show the approximate fingerprints property, it suffices to show that the fraction of concepts in T_n that classify α as L does is superpolynomially small. As in Lemma 23, let $m=\sqrt{n\ln p}$. If $L(\alpha)$ is 1, the number of concepts in T_n that classify α as L does is at most $\binom{m}{q}(n-q)!/((q!)^{q-1}(q-1)!)$. If $L(\alpha)$ is 0, the number of concepts in T_n that classify α as L does is at most $\binom{m}{q}(n-q)!/([(q-1)!]^q q!)$. In either case, the fraction of concepts in T_n that classify α as L does is at most $\binom{m}{q}((n-q)!q^q)/n!$, which (after plugging in the values of m and q) is less than $(\sqrt{nc\ln n})^{\sqrt{n}}/(\sqrt{(n)})!(\sqrt{n})^{\sqrt{n}}/(n-\sqrt{n})^{\sqrt{n}}$. This quantity can be seen to be superpolynomially small in n. □

Corollary 25. *The class of monotone term decision lists is not learnable with equivalence queries alone.*

Proof. Follows from Theorem 24 and Angluin's theorem [2] on the approximate fingerprints property. □

5.2. A learnable subclass of monotone term decision lists

Let disj-MDL be the subclass of Boolean functions representable as monotone term decision lists in which no two terms of different output-value share a variable. Equivalently, the set of relevant variables of a disj-MDL function may be partitioned into

1. $H \leftarrow (\text{True},\ \text{MQ}(0^n))$
2. **while** $\text{EQ}(H) = \text{“No”}$ **do**
3. Let α be the counterexample
4. $\alpha \leftarrow$ Minimize(H, α)
5. Let (t, c) be the node activated by α in H
6. Let t' be the term such that $vars(t') = ones(\alpha) - vars(t)$
7. $H \leftarrow \text{Sort}(\langle (t', \bar{c}), H \rangle)$
8. **end-while**

Fig. 1. The algorithm for learning disj-MDL.

two sets – the set of variables which appear in nodes of output value 0 and the set of variables which appear in nodes of output value 1. Note that this class includes all monotone functions.

The class disj-MDL is not learnable with membership queries alone because of the fact that singletons can be represented using a disj-MDL representation of at most n nodes. Also, it follows from the proof of Theorem 24 that disj-MDL cannot be learned with proper equivalence queries (in fact, even if the queries are general monotone term decision lists) because the target class T_n used in the proof is contained in disj-MDL. Therefore, if disj-MDL are properly and exactly learnable at all, one must use both types of queries.

Theorem 26. *The class of disj-MDL is exactly and properly learnable with membership and equivalence queries. More precisely, any Boolean function over n variables representable as an m-term disj-MDL can be learned exactly using* $m + 1$ *proper equivalence queries,* $O(m^2 + mn)$ *membership queries, and* $O(m^2 + mn)$ *time.*

Proof. Let H^* be a minimal disj-MDL representing the target concept. Let n be the number of variables of the target function and m the size of H^*.

Consider the algorithm in Figs. 1 and 2.

We prove the correctness of the algorithm using the following invariant:

Invariant.

- For every node (t, b) in H, (t, b) is a node in H^*.
- For every pair of nodes of different output values (t_1, b) and $(t_2, \bar{b})$ in H,

  (t_1, b) appears before $(t_2, \bar{b})$ in H if and only if (t_1, b) appears before $(t_2, \bar{b})$ in H^*.

The invariant is initially satisfied when the *while* loop is entered since we ask a membership query on 0^n to get a hypothesis with only the default node. Assume that the invariant is satisfied at the beginning of some iteration of the *while* loop. We use the following claims to show that the invariant is true at the end of the current iteration of the *while* loop.

function Minimize(H, α):
// Let (t,b) be the node activated by α in H.
// Minimize returns a *reduced* counterexample, i.e., an assignment α'
// such that $H(\alpha') \neq H^*(\alpha')$, α' also activates (t,b) in H,
// and for all variables $v \in ones(\alpha') - vars(t)$, $\alpha'_{v\leftarrow 0}$ activates (t,b) in H^*.

1. Let (t,b) be the node activated by α in H
2. Let $V = \{v_1, \ldots, v_n\}$ be the set of variables
3. $\alpha' \leftarrow \alpha$
4. **repeat** 2 times
5. **for** $j \leftarrow 1$ **to** n **do**
6. **if** $v_j \in ones(\alpha') - vars(t)$ and $\mathrm{MQ}(\alpha_{v_j\leftarrow 0}) = \overline{b}$ **then**
7. $\alpha' \leftarrow \alpha'_{v_j\leftarrow 0}$
8. **endif**
9. **end for**
10. **endrepeat**
11. **return** α'

Fig. 2. The function Minimize.

Claim 27. *After step* 4 *is executed in the algorithm in Fig.* 1, *there exist nodes* $(t,c) \in H$ *and* $(t',\overline{c}) \in H^*$, *activated both by* α *in* H *and* H^*, *respectively*, *such that*
1. $ones(\alpha) = vars(t) \cup vars(t')$
2. $(t',\overline{c})$ *appears before* (t,c) *in* H^*
3. $(t',\overline{c})$ *does not appear in* H

Proof of Claim 27. Since α is a counterexample for H, α must activate some node $(t',\overline{c})$ in H^* and some node (t,c) in H.

We prove the claim, item by item:

1. Clearly, $vars(t) \cup vars(t') \subseteq ones(\alpha)$ since α activates both $(t',\overline{c})$ and (t,c). Now we show that Minimize (see Fig. 2) guarantees that there is no variable in $ones(\alpha) - (vars(t) \cup vars(t'))$. Consider the set $N_{\alpha'}$ of nodes of H^* that appear before (t,c) and whose terms are satisfied by α'. We show that at step 11, $|N_{\alpha'}| = 1$.
   Let V_a be the variables involved in nodes with output value a. After the first round of the *for* loop in step 4, all variables from $V_c - vars(t)$ have been flipped to 0 and therefore no node of output value c is in $N_{\alpha'}$ any more. The second round deletes all but one node of output value $\overline{c}$ since if there were two different nodes, $(r,\overline{c})$ and $(s,\overline{c})$ in $N_{\alpha'}$, some variable in the symmetric difference of $vars(r)$ and $vars(s)$ could be set to 0. This proves that $|N_{\alpha'}| \leqslant 1$, and the fact that α' is still a counterexample implies the equality. This node that remains in $N_{\alpha'}$ is precisely $(t',\overline{c})$ so the second round has also set to 0 all variables in $V_{\overline{c}} - vars(t')$. This implies that $ones(\alpha') = vars(t) \cup vars(t')$.

2. The first part of the invariant ensures that (t,c) is also a node in H^*. If $(t',\overline{c})$ appears after (t,c) in H^*, α would not activate $(t',\overline{c})$ in H^*.
3. If $(t',\overline{c})$ were in H, by the second part of the invariant, it would appear before (t,c) and α would have not activated (t,c) in H. □

Claim 27 implies that the node $(t',\overline{c})$ computed in step 6 is a new node of H^* not yet placed in H. In order to maintain the invariant, we have to prove that we can construct a new hypothesis disj-MDL representation which includes all the old nodes of H, includes the new node $(t',\overline{c})$, and maintains the order required by the second part of the invariant.

Claim 28 provides a tool for this purpose.

Claim 28. *Let (r,d) and $(s,\overline{d})$ be two nodes in H^* and let $\alpha_{r,s}$ be an assignment such that $ones(\alpha_{r,s}) = vars(r) \cup vars(s)$. Then (r,d) appears before $(s,\overline{d})$ in H^* if and only if $H^*(\alpha_{r,s}) = d$.*

Proof of Claim 28. We prove the claim by showing that (r,d) appears before $(s,\overline{d})$ in H^* if and only if $\alpha_{r,s}$ activates (r,d) in H^*.

Now suppose, to the contrary, that (r,d) appears before $(s,\overline{d})$ in H^*, but $\alpha_{r,s}$ activates some node (u,e), prior to (r,d). If $e = d$ then $vars(u) \subseteq vars(r)$ because H^* is a disj-MDL representation but this renders (r,d) redundant, contradicting the minimality of H^*. If $e \neq d$ then a similar argument shows that $(s,\overline{d})$ is redundant, yielding the same contradiction. Conversely, if $\alpha_{r,s}$ activates (r,d) it must be because (r,d) appears before $(s,\overline{d})$.

Now if $\alpha_{r,s}$ activates (r,d) in H^* then clearly $H^*(\alpha_{r,s}) = d$ and conversely if $H^*(\alpha_{r,s}) = d$ it must because $\alpha_{r,s}$ activates (r,d) in H^* since we have ruled out the possibility of activating some other node. □

Claim 28 implies that we can find out efficiently the relative order in H^* of two nodes of different output value and that this order is always defined. The procedure in step 7 (Sort) simply obtains this partial order using membership queries on all assignments $\alpha_{r,s}$ where (r,d) and $(s,\overline{d})$ are nodes in H and then outputs a disj-MDL by a topological sort of the partial order. Clearly, this satisfies the invariant at the end of the current iteration of the *while* loop.

After each iteration of the *while* loop, H contains a new node of H^* in the correct relative order of nodes of different output values. To finish the proof of correctness, it remains only to show that once H contains all the terms from H^* the equivalence query must be answered positively. Suppose that this is not the case. Then there exists a counterexample α that activates some node (t,a) in H and some node $(r,\overline{a})$ in H^*. The invariant implies that (t,a) is placed before $(r,\overline{a})$ in H^* but this contradicts the fact that α activates $(r,\overline{a})$ in H^* instead of (t,a).

From the above, the number of equivalence queries is bounded by $m+1$. Note that Sort needs only to find out the relationship of the current new node $(t',\overline{c})$ with all the existing nodes of opposite value since the relationship of all other nodes of opposite

value will be known at this stage. This implies that Sort needs to make at most $\binom{m}{2}$ membership queries overall. The other place where membership queries are used is in the function Minimize. Each invocation of Minimize uses at most $2n$ membership queries, and so the total number of membership queries used in Minimize is at most $(m+1)2n$.

Each invocation of Sort needs at most $O(m)$ time to place correctly the new node and rearrange the other nodes. This is because the new node $(t', \bar{c})$ can affect only a block of nodes of output value c, splitting them into two new blocks: the nodes that must appear before and the nodes that must appear after. Each invocation of Minimize takes at most $O(n)$ time. Since these are the bottleneck steps in the at most $m+1$ iterations of the *while* loop, this results in the claimed timebound of $O(m^2+mn)$ for the entire algorithm. □

The following corollary is immediate from the proof of Theorem 26:

Corollary 29. *A minimal disj-MDL representation of a Boolean function is unique modulo permutation of nodes within a block.*

Proof. This follows directly from the invariant used in the algorithm. □

It remains to relate the representation size of disj-MDL with the size using the monotone term decision list representation. We prove that the disj-MDL size is optimal with respect to the monotone term decision list size.

Let $|f|_{dmdl}$ be the number of terms of the smallest disj-MDL that represents f, if f is so representable.

Theorem 30. *For any Boolean function f representable as a disj-MDL, $|f|_{dmdl} = |f|_{mdl}$.*

Proof. Clearly, $|f|_{dmdl} \geqslant |f|_{mdl}$.

Let L be a minimal disj-MDL representation of f and let M be any monotone term decision list representing also f. To show that $|f|_{dmdl} \leqslant |f|_{mdl}$, we build a set A of $|f|_{dmdl}$ assignments such that no pair of assignments in A can activate the same node in M. For any node (t,b) in L, let $(s_t, \bar{b})$ be the first node of different output value that appears after (t,b) in L. Let α_{s_t} and α_{t,s_t} be the assignments defined by $ones(\alpha_{s_t}) = vars(s_t)$ and $ones(\alpha_{t,s_t}) = vars(t) \cup vars(s_t)$ and let $A = \{\alpha_{t,s_t} : (t,b) \in L\}$. We need the following claim.

Claim 31. *For every assignment β such that $\alpha_{t,s_t} > \beta \geqslant \alpha_{s_t}$, β* activates $(s_t, \bar{b})$.

Proof of Claim 31. By Claim 28, the assignment α_{t,s_t} activates the node (t,b). Since at least one of the variables in $vars(t)$ is set to 0 in β, β must activate some node after (t,b). Suppose, for the sake of contradiction, that β activates (r,b) a node after (t,b) but before $(s_t, \bar{b})$ (i.e., β does not activate $(s_t, \bar{b})$). Since $vars(r) \subseteq ones(\alpha_{t,s_t})$ and

f is a disj-MDL $vars(r) \subseteq vars(t)$. Now, using that there is no node of output value $\bar{b}$ between (t,b) and (r,b), the node (t,b) can be deleted without changing the function computed by the list, contradicting the minimality of L. □

Claim 31 implies that for every term t in a node of L, α_{t,s_t} must activate a node (r,b) in M such that $vars(t) \subseteq vars(r)$ (otherwise some assignment v such that $\alpha_{t,s_t} > v \geqslant \alpha_{s_t}$ would also activate (r,b) and M would not be equivalent to L). We are now ready to show that every assignment $\alpha_{t,s_t} \in A$ activates a distinct node in M, thereby proving the theorem.

Suppose to the contrary that there are two assignments α_{t,s_t} and $\alpha_{t',s'_{t'}}$ that activate the same node (r,b) in M. Observe that the output value of the terms t and t' coincide and the same happens with s_t and $s_{t'}$. Using the same argument as above, $vars(t') \subseteq vars(r)$ and therefore

$$(vars(t) \cup vars(t')) \subseteq vars(r) \subseteq (ones(\alpha_{t,s_t}) \cap ones(\alpha_{t',s'_{t'}})).$$

Since L is minimal, $t \neq t'$ and consequently there is some variable x in the symmetric difference of $vars(t)$ and $vars(t')$. Now x cannot be in $vars(s_t) \cap vars(s'_{t'})$ because L is a disj-MDL representation and a variable x that comes from a b-node (t or t') cannot appear in a $\bar{b}$-node s_t or $s_{t'}$. However, this implies that $vars(t) \cup vars(t')$ is a proper subset of $vars(t) \cap vars(t')$, a contradiction. □

6. Conclusions

We have presented a class of decision lists in which the terms are restricted to being monotone. In spite of this restriction, several desirable properties of general decision lists, such as succinctness and universality, are preserved. In addition, monotone term decision lists are amenable to efficient operations such as equivalent testing, rendering a representation irredundant, satisfiability/tautology testing, etc., which are intractable for other universal representation classes.

Some subclasses of monotone term decision lists are learnable: k-term monotone decision lists are properly and exactly learnable with membership queries alone, for any constant k, while they can neither be properly PAC-learned nor learned with equivalence queries alone. The subclass of disjoint-MDL is properly learnable with equivalence and membership queries while neither type of query alone will suffice.

The main question that this work leaves open is the learnability of general monotone term decision lists. It would be interesting to know if the class is at least predictable with membership queries.

Other open questions have to do with the representation class of monotone term decision lists:

- Is there a polynomial time algorithm for minimizing a given monotone term decision list?
- Is there a polynomial time algorithm for truth-table minimization of a monotone term decision list?

Given that monotone term decision lists are as good or better than decision trees when one considers representation size and other issues, it would be of interest to see how a practical inference engine based on monotone term decision lists compares with popular decision-tree-based learners.

Acknowledgements

The authors wish to thank José L. Balcázar, Ricard Gavaldà and Jorge Castro for suggesting the class and participating in the discussion. A preliminary version, with a subset of the results presented here, appeared in Eurocolt'97 [15].

References

[1] D. Angluin, Queries and concept learning, Machine Learning 2 (1988) 319–342.
[2] D. Angluin, Negative results for equivalence queries, Machine Learning 5 (1990) 121–150.
[3] D. Angluin, L. Hellerstein, M. Karpinski, Learning read-once formulas with queries, J. ACM 40 (1) (1993) 185–210.
[4] D. Angluin, M. Kriķis, Learning with malicious membership queries and exceptions, Proc. 10th Ann. Conf. on computational Learning Theory, ACM Press, New York, NY, 1997, pp. 285–297.
[5] M. Anthony, N. Biggs, Computational Learning Theory: An Introduction, Cambridge University Press, Cambridge, 1992.
[6] M. Anthony, G. Brightwell, J. Shawe-Taylor, On specifying Boolean functions by labelled examples, Discrete Appl. Math. 61 (1995) 1–25.
[7] M. Blum, A. Chandra, M. Wegman, Equivalence of free Boolean graphs can be decided probabilistically in polynomial time, Inform. Process. Lett. 10 (1980) 80–82.
[8] R. Board, L. Pitt, On the necessity of Occam algorithms, Theoret. Comput. Sci. 100 (1992) 157–184.
[9] R.E. Bryant, Graph based algorithms for Boolean function manipulation, IEEE Trans. Comput. C-35 (1986) 677–691.
[10] N. Bshouty, Simple learning algorithms using divide and conquer, Proc. 8th Ann. Conf. on Computational Learning Theory, 1995, pp. 447–453.
[11] J. Castro, J.L. Balcázar, Simple PAC Learning of Simple Decision Lists, Proc. 6th Internat. Workshop on Algorithmic Learning Theory (ALT'95), Lecture Notes in Artificial Intelligence, Vol. 997, Springer, Berlin, 1995, pp. 239–250.
[12] A. Ehrenfeucht, D. Haussler, Learning decision trees from random examples, Inform. and Comput. 82 (1989) 231–246.
[13] M. Fredman, L. Khachiyan, On the complexity of dualization of monotone disjunctive normal form, J. Algorithms 21 (1996) 618–628.
[14] S. Goldman, M. Kearns, On the complexity of teaching, Proc. 4th Ann. Workshop on Computational Learning Theory, 1991, pp. 303–314.
[15] D. Guijarro, V. Lavín, V. Raghavan, Learning monotone term decision lists, Proc. 3rd European Conf. on Computational Learning Theory (Eurocolt'97), Lecture Notes in Artificial Intelligence, Vol. 1208, Springer, Berlin, 1997, pp. 16–26.
[16] T. Hancock, T. Jiang, M. Li, J. Tromp, Lower bounds on learning decision lists and trees, Inform. and Comput. 126 (2) (1996) 114–122.
[17] L. Hellerstein, K. Pillaipakkamnatt, V. Raghavan, D. Wilkins, How many queries are needed to learn?, J. ACM 43 (5) (1996) 840–862.
[18] M. Kearns, U. Vazirani, An Introduction to Computational Learning Theory, MIT Press, Cambridge, MA, 1994.
[19] E. Kushilevitz, A simple algorithm for learning $O(\log n)$-Term DNF, Proc. 9th Annual Conf. on Computational Learning Theory, 1996, pp. 266–269.

[20] M. Li, P. Vitányi, Learning simple concepts under simple distributions, SIAM J. Comput. 20 (1991) 911–935.
[21] R. Lipton, The reachability problem requires exponential space, Research Report 62, Department of Computer Science, Yale University, New Haven, Connecticut, January 1976.
[22] B.K. Natarajan, Machine Learning: A Theoretical Approach, Morgan Kaufmann, San Mateo, CA, 1991.
[23] J.L. Peterson, Petri nets, Comput. Surveys 9 (3) (1977) 223–252.
[24] L. Pitt, L. Valiant, Computational limitations on learning from examples, J. Assoc. Comput. Mach. 35 (4) (1988) 965–984.
[25] R. Rivest, Learning decision lists, Mach. Learning 2 (1987) 229–246.
[26] H.U. Simon, Learning decision lists and trees with equivalence queries, Proc. 2nd European Conf. EUROCOLT, 1995, pp. 322–336.
[27] E. Takimoto, Y. Sakai, A. Maruoka, Learnability of Exclusive- Or Expansion Based on Monotone DNF Formulas, Proc. 7th Internat. Workshop on Algorithmic Learning Theory (ALT'96), Lecture Notes on Artificial Intelligence, Springer, Berlin, Vol. 1160, 1996, pp. 12–25.
[28] L. Valiant, A theory of the learnable, Commun. ACM 27 (11) (1984) 1134–1142.
[29] O. Watanabe, A Formal Study of Learning via Queries, Proc. 17th Internat. Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, Vol. 443, Springer, Berlin, 1990, pp. 139–152.
[30] I. Wegener, in: The Complexity of Boolean Functions, Wiley-Teubner Series in Computer Science, Teubner, Stuttgart/Wiley, Chichester, 1987.